A Secure Cloud Authentication Model Leveraging Hash-Based Access Tokens and Cookie Validation

EBRAHIM RAHAMAN SONCHAY

Bachelor of Science

DAFFODIL INTERNATIONAL UNIVERSITY

APPROVAL

This thesis titled on "**A Secure Cloud Authentication Model Leveraging Hash-Based Access Tokens and Cookie Validation**", submitted by **Ebrahim Rahaman Sonchay (ID: 221-35-975)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Dr. A. H. M. Saifullah Sadi
Professor
Department of Software Engineering
Faculty of Science and Information Technology Daffodil International University

Chairman

Dr. Rubaiyat Islam
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 1

Dr. Md. Abdul Kader
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 2

Nuruzzaman Faruqui
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 3

Md. Mostafiz Khan
Managing Director
Tecognize Solutions Limited

External Examiner

DAFFODIL INTERNATIONAL UNIVERSITY

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	:	Ebrahim Rahaman Sonchay
Date of Birth	:	11/12/2001
Title	:	A Secure Cloud Authentication Model Leveraging Hash-Based Access Tokens and Cookie Validation
Academic Session	:	

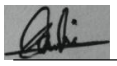
I declare that this thesis is classified as:

☐	CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997)*
☐	RESTRICTED	(Contains restricted information as specified by the organization where research was done)*
☐	OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Daffodil International University reserves the following rights:

1. The Thesis is the Property of Daffodil International University.
2. The Library of Daffodil International University has the right to make copies of the thesis for the purpose of research only.
3. The Library of Daffodil International University has the right to make copies of the thesis for academic exchange.

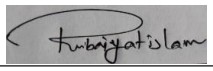
Certified by:



(Student's Signature)

221-35-975

Student ID
Date: 21-12-2025



(Supervisor's Signature)

Dr. Rubaiyat Islam
Name of Supervisor
Date: 21-12-2025

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Daffodil International University,
Daffodil Smart City,
Ashulia.Dhaka,Bangladesh

Dear Sir,

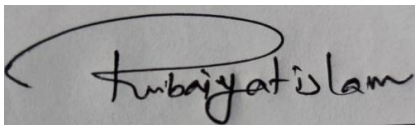
CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name	Ebrahim Rahaman Sonchay
Thesis Title	A Secure Cloud Authentication Model Leveraging Hash-Based Access Tokens and Cookie Validation
Reasons	(i) The thesis contains sensitive technical details and system-level implementation information that could be misused if made publicly accessible. (ii) The research includes original methodologies, algorithms, and experimental data that are currently intended for future publication, patent filing, or further academic work.

Thank you.

Yours faithfully,



(Supervisor's Signature)

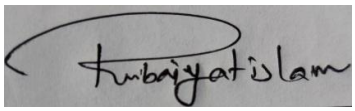
Date: 21-12-2025

Stamp:

Note: This letter should be written by the supervisor and addressed to the Librarian, *Daffodil International University* with its copy attached to the thesis.

SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and in my opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Bachelor of Science of Science.



(Supervisor's Signature)

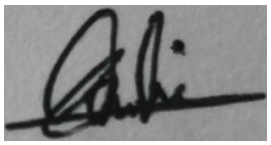
Full Name : Dr. Rubaiyat Islam

Position : Associate Professor

Date : 21 December2025

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Daffodil International University or any other institution.



(Student's Signature)

Full Name : Ebrahim Rahaman Sonchay

ID Number : 221-35-975

Date : 21 December 2025

A Secure Cloud Authentication Model Leveraging Hash-Based Access Tokens and Cookie Validation

EBRAHIM RAHAMAN SONCHAY

Thesis submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Science

Department of Software Engineering (Major in Cyber Security)

DAFFODIL INTERNATIONAL UNIVERSITY

NOVEMBER 2024

ACKNOWLEDGEMENTS

At the very beginning, I would like to express my heartfelt gratitude to the Almighty Allah. His countless blessings, guidance, and mercy have supported me throughout every stage of my thesis journey. Without His grace, it would not have been possible to complete this work successfully.

I am deeply thankful to my supervisor, **Dr. Rubaiyat Islam**, Associate Professor, Department of Software Engineering, Daffodil International University. His continuous support, thoughtful guidance, and encouragement have shaped this research from start to finish. He was always available whenever I needed help, and his patience and expertise made this work possible.

Software Engineering, Daffodil International University. Their collaboration,
Motivation and academic support have always formed the foundation of my educational pursuit.

I would like to add my comments on one of the objectives being pursued by the LBA by stating that I would like to point out one of the objectives being sought by including the following:

I also would like to thankmy friends and associates who shared the fun, support, and disagreements in the course of this process.

discussions, and encouragement that helped during the course and this thesis process.
Most importantly, I am grateful to my parents. Their unconditional love and infinite
The biggest factor that has acted in my favor is the encouragement, support, and motivation that I have always received. Everything that I
what I attain is through their sacrifice and prayers.

DEDICATION

This thesis is dedicated to my parents, who have always supported me and encouraged me to work on this project and complete my sacrifices have remained the driving factor in my pursuit of education. Their love, patience,
support and belief in my abilities. All of whom have helped mold me into the person I am today. For which I am very grateful for everything.
they have done for me.

ABSTRACT

As the cloud computing trend becomes increasingly popular, modern web applications are increasingly likely to be based on cloud computing services
rely on distributed systems which are scalable and flexible. Rapid evolution of technology has also influenced the development of the industry. All
Something immediate Almost overnight Inevitable But with this speed, security
threats with the objective of authentication and session management have been on the increase. Threats such as
even now the session hijacking, cookie stealing, token forgery, and replay attacks remain critical problems
“threats to the security and integrity of cloud services. Traditional methods—such
“as passwords, cookies, JWTs, and browser fingerprinting—are inadequate to provide”
strong security within a multi-tenant, device-diverse environment, token replay
vulnerabilities leak of sensitive data
leak of sensitive data
leak of sensitive data
techniques.

These challenges can be overcome by presenting a secure cloud authentication model in this paper.
Based on hash-based verification for the access and refresh tokens, by combining with
client-specific attributes, including IP address, browser meta data, and device IDs. The system
ensures that every authentication session is bound to the place it originated from. It
dodges the typical JWTs with sensitive information of user, protecting
Data from stolen token

In the core of our system, there is a two-level verification process involving the client-side hashing and server-side querying. It gives superior protection against the following:
cookie theft threats, replay attacks, and cross-site request forgeries. If a token
becomes compromised, it cannot be connected with any other device or network, hence
by greatly reducing abuse.

It also offers secure and transparent continuity for sessions using the secured renewal of tokens. It has some cost associated with computations and does not perform well with extensively fluctuating conditions for IPs. It achieves a good tradeoff between security and usability.

In this paper, a new system that ensures increased privacy, integrity, and trustworthiness within clouds has been presented through a viable authenticating system that lessens dependence on token approaches, thus opening new avenues in the future, such as cache optimization, cryptography, and anomaly detectors using AI.

Keywords: Secure Cloud Authentication, Hash-Based Token Validation, Cookie Theft Prevention, JWT Security, Dual-Layer Verification

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 Background

Today in the rapidly changing digital environment, secure authentication has become a crucial part of
cloud services, cloud computing, and cloud applications. As the volume of data continues to rise, it has become challenging for businesses to
applications that find themselves migrated to cloud solutions, and the number of such applications continues to increase, organizations
and users alike are increasingly turning to third-party infrastructure in attempt to enable the storage and processing of sensitive information. Such a move presents the potential for new security threats, as attackers increasingly exploit vulnerabilities in authentication processes. As stated in the given article, Authentication Header (
and session management systems to break into. If we continue to use these tools, not only the
above attacks can happen increasingly and increasingly often and with greater and greater sophistication to steal
in silence steals the valuable user's data, such as session hijacking, stealing cookies, or token misuse, but
It is also the point at which most issues regarding the privacy of the end-user might also be kept hidden.

1.2 Limitations of Traditional Authentication

"Basic/Digest Client-Server Authentication and Cookie-Based Sessions," the methods used for historic authentication, while working in a simpler web environment, no longer meet the required standards. However, the point is, in the modern world of cloud native systems, "in the present day, it does not suffice." Passwords can be "guessed, stolen, or reused among multiple services and cookies remain vulnerable to theft, capture, and replay attacks." There are issues with more advanced authentication methods like "JSON Web Tokens (JWT); for example," and "Security Assertion Markup Language-Based SAML."
browser fingerprinting:

- You inadvertently leak user information if you send the JWT payload in clear text.
- Browser based finger/prints may be unreliable, spoofed, or vary in dynamic network settings.

These deficiencies highlight the requirement for a more secure, realistic, and convenient authentication method that is capable of withstanding today's threat models while maintaining good usability.

1.3 Proposed Solution

This paper adds a new approach related to a security cloud authentication model with improved session integrity by tying the access token and refresh token to a client environment. The approach involves hashing client-specified details, such as IP address, browser information, and device information, in the verification process. By not transmitting sensitive details in token responses, combined with using signatures that verify tokens to be in a specific environment, it prevents most risks posed by tampered tokens when usable in a new device and network environment.

A key component of the proposed design is its dual-layer verification mechanism. This includes:

- Verification of hashes on the client side to ensure token authenticity before it is sent over.
- Server-side validation of the database for tampered or unauthorized use of tokens.

It improves security with this layered approach, maintaining seamless usability through secure token renewal for the legitimate user.

1.4 Research Contributions

The primary contributions of this research are as follows:

- **Token protection:** Some security is enforced on access and refresh tokens by cryptographic hashing in combination with client-specific environment properties.
- **Prevention of cookie / token hijacking:** This measure prevents the utilization of hijacked cookies or tokens on alternative networks or devices.
- **Improved privacy:** The JWT payloads no longer contain sensitive user information, implying that less sensitive information is encoded in the URLs.
- **His and Her validation:** The combination of client-side hashing with server-side checks yields defense against token forgery, replay attack as well as unauthorized access.

1.5 Summery

Finally, in this context, we conclude that we have given a new, practical, and private solution for the ever-changing threats found in cloud-based authentication. With the connection of certain authentication sessions with certain client contexts, the proposed model gives enhanced security, as well as trust for cloud-based applications, than was previously possible with older cloud-authentication mechanisms.

CHAPTER 2

RELATED WORK

2.1 Overview of Web Authentication Challenges

The strength of the web authentication system has been substantial in consuming academic research in the last decade. Although much progress has been achieved in this area, some common techniques such as cookie-based sessions, JSON Web Token (JWT), and browser fingerprinting are still vulnerable, especially in cloud-centric environments.
These include session hijacking, token replay attack/privacy violation, among others, which are causing a serious concern and indicate a need to adopt effective measures against such risks.

2.2 Limitations of Existing Authentication Techniques

Several studies have identified the security shortcomings of current authentication approaches:

2.2.1 Vulnerabilities in Browser Fingerprinting

Fouad et al. (2020) found that browser fingerprinting is not immune to sophisticated attacks, where attackers are able to restore previously deleted cookies and keep unauthorized access without user consent [3]. This extended session hijacking is a significant privacy threat.

Yan et al. (2022) have added that while fingerprinting can improve authentication accuracy, it remains vulnerable to instability, environmental inconsistency and phishing as well as man-in-the-middle (MitM) attacks [9].

2.2.2 Security Issues in JWT-Based Authentication

Jang (2021) focused on the serious vulnerabilities of JWT systems, and pointed out that sensitive information is frequently stored in token payloads [5]. An intercepted JWT that has not been tampered with can be replaced, impersonated, or modified since they have predictable structures and their content is base64 encoded.

2.2.3 Limitations in Cookie-Based Sessions

Conventional cookie-based sessions are still exposed to theft, replay and cross site scripting (XSS) attacks. Research has uniformly affirmed the inadequacy for using cookies to protect authentication sessions across distributed clouds.

2.2.4 Overall Observations

Common limitations While reviewing the published literature, several common limitations are found across these and other studies:

- Prevent unauthorized token replay was not possible
- Leaks in token payloads for sensitive user information
- Low session server/client authentication binding

Also, this informs that the solutions currently given are providing a low level of security atisfaction, meaning that they do not provide you with an “agree to disagree” model.

2.3 Literature Summary

Table I presents an overview on the contributions and gaps in existing studies conducted on secure Authentication systems in the clouds.Table I: Summary of Key Authentication Techniques

Ref	Method	Contribution	Limitation
[1]	TFHV	MitM protection	Complex setup
[2]	MCU-Token	IoT ML defense	Hardware dependent
[3]	Fingerprint detect	Cookie respawn prevention	Limited features
[4]	OIDC timing	Risk profiling	Network dependent
[5]	JWT analysis	Base64 encoding fix	HTTP limits
[6]	Multi-tier JWT + crypto + OTP	Multi-layer security	High complexity
[7]	Triple-ID	Dynamic hash mechanism	Algorithm complexity
[9]	Cache channel HW fingerprint	Hardware fingerprinting	CPU-specific limits

[10]	ML context behavioral model	Context-based anomaly detection	High training requirements
[15]	JWT evaluation RFC 8725 study	Standards compliance	Implementation gaps
[22]	Device binding	Hardware-session binding	Limited support
[23]	Threat monitor	Session detection	Scaling issues

2.4 Research Gap Analysis

A systematic analysis of the literature identifies several critical gaps:

2.4.1 Lack of Integrated Multi-Layer Defense

Prior art does propose certain security methods, but few of them bring together several complementary solutions — e.g., hashing, environment validation and token It means protection within one model.

2.4.2 Complexity vs. Practicality Trade-Off

More complex systems like multi-level JWT frameworks or ML-based authentication are more secure, and much trickier and complex to implement – so the past

2.4.3 Fragmented Hardware-Software Integration

Some of the studies focus on hardware-based techniques, including device binding or It is alternatively known as hardware fingerprinting. However, these methods are not cross-platform and hard standardizing.

2.4.4 Standards vs. Implementation Gap

Evidence of this are the common issues that researchers have found between standards (as RFC 8725 for JWT) and their real world use. This space opens the room to configurations and vulnerabilities that can be exploited.

2.4.5 Limited Adaptability to Dynamic Threats

However, to the best of our knowledge, existing authentication systems do not provide adaptive mechanisms support each other in identifying with difficult to detect token replay (Hacking) or session hijacking / new phishing forms.

2.5 Positioning of the Proposed Solution

The secure cloud authentication model presented here fills the gaps that were identified by providing a comprehensive method:

2.5.1 Hash-Based Token Validation

It uses the combination of the cryptographic hashing function and the token verification technique. This helps thwart the issues associated with replay

2.5.2 Client-Environment Binding

Associates access and refresh tokens to specific client features such as the IP address, the browser, or the device identifier to avoid misuse of the token by other devices or geographic locations.

2.5.3 Dual-Layer Verification

Combines:

- Client-side generated hash to verify token's authenticity before the transmission parties.
- Server-validation of data integrity in the database, serving to detect unauthorized intrusions and tampering

2.5.4 Practical and Privacy-Preserving Design

Snippet 7: Ideal Sensitive payload transforms into a "usable but safe" token payload sensitive data is stripped from the token payloads - minimizing exposure risk, while keeping it usable through a safe token refreshing process.

2.6 Summary

The review of literature shows that several mechanisms for authentication are available; however, they are not an integrated client-side receptor-privacy mechanism suitable to current cloud environments. This gap is addressed by combining Hash based authentication, environment binding, and a multiple layered security architecture to provide usability and theoretical sound secure framework.

CHAPTER 3

TERMINOLOGY AND BACKGROUND CONCEPTS

3.1 Authentication Terminology

3.1.1 JSON Web Token (JWT)

A JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. A JWT is comprised of 3 parts:

- Header: Type of token and signing algorithm.
- Payload: Includes user claims, metadata and session data.
- Signature: Indicates the token has not been tampered with.

Example format: eyJhbGci. eyJzdWIi. SflKxwRJ

By the way, JWTs support Stateless Authentication, meaning that our server does not need to keep sessions. However, placing sensitive data in the payload exposes users to threats such as token theft, replay attacks and forgery [5].

3.1.2 HttpOnly Cookies

The HttpOnly flag prevents cookies from being accessed by Javascript running in the browser, which is a defense against cookie-based XSS token theft. Yet, browser fingerprinting research has demonstrated that even this degree of protection can be defeated by re-creating the cookies [3], so persistent session hijacking and cookie respawn is possible.

3.1.3 SameSite Cookie Attribute

As a means to protect against CSRF attacks, the SameSite attribute specifies when cookies will be sent:

- Strict: Only sent with same-site requests.
- Lax: Sent with only top level navigation (default for modern browsers).
- None : When sent with any request, but secured with the Secure flag.

3.1.4 Bearer Tokens

The bearer has access to anything the token grants its subset access to. Example:

```
Authorization: Bearer <token>
```

If intercepted, attackers would have the ability to impersonate users. Bearer token implementations based on JWT has known vulnerabilities [17].

3.1.5 Session Tokens

Session tokens serve as unique keys to the persisted session data kept on the server. These vouchers can be easily stolen if not properly secured. Research demonstrates that hijacking session cookies can even evade multi-factor authentication (MFA) [23].

3.1.6 Access Tokens and Refresh Tokens

- **Access Tokens:** Small (15–30 minutes), representing the user and their permissions.
- **Refresh Tokens:** Long-lived (sv days/weeks), only used for acquiring a new access token. These should be kept securely and revocable. In [1] we already presented hybrid encryption schemes to secure refresh tokens.

3.2 Cryptographic Concepts

3.2.1 SHA-256

A cryptographic function on normal distribution whose digest is always 256 bits. It is deterministic, non-reversible and avalanche effect. Used often for hashing tokens/ deviceina (devices fingerprint).

3.2.2 HMAC

A cipher is a combined operation of the key and hash functions: vs Hash-based Message Authentication Code (HMAC), which applies a cryptographic hash over a secret key:

$$HMAC = Hash((key \oplus opad) \,||\, Hash((key \oplus ipad) \,||\, message))$$

HMAC can defend against integrity attack, but it is vulnerable to weak keys [19].

3.2.3 Salt

Random data padded to the input before hashing for varying hash output.

3.2.4 PBKDF2

It is a procedural method to derive the key used as input by iterating the HMAC construction. Widely used in secure matching on keys.

3.2.5 Constant-Time Comparison

Constant time comparison of two numerical values to avoid timing attacks. It masks the activities and hides internal values from attackers using timing through differences in execution [4].

3.2.6 Hash Chain

In a chain of dependent hashes for authenticating and forward secure refresh tokens to maintain one-way continuity, so reusing refresh token isn't possible.

3.3 Attack Vectors

3.3.1 Session Hijacking

Infiltrating with session ID stolen by packet sniffing, session fixation, trojan or XSS. Not so with (^mac,^big]) device-bound fingerprints [21].

3.3.2 Cookie Theft

Happens via malware, MITM or XSS attacks. Verification using hashs makes it impossible to reuse stolen cookies acrosse environments [22].

3.3.3 Session Fixation

Session fixation Attacker sets the session ID before authentication, and is then able to hijack the victim's session after the user authenticates themselves Phrased differently, an attacker can create a valid identity for themself on a website by forcing you to use their session as your own. mitigated by regenerating session IDs=[], using always new IDs immediately after new login.

3.3.4 Token Replay Attack

Re-Creating monitors using tokens intercepted during user impersonation. Alleviated with timestamp, nonce, and one-time token.

3.3.5 Man-in-the-Middle (MITM)

Cross-linking of communication path or tampering. Defences include HTTPS, HSTS and certificate pinning.

3.3.6 Cross-Site Scripting (XSS)

Access through running malicious scripts to extract cookies or control the session. HttpOnly cookies provide some mitigation [20].

3.3.7 Cross-Site Request Forgery (CSRF)

Forces users to take an unwanted action while authenticated. Mitigations exist such as SameSite cookies and CSRF tokens.

3.3.8 Timing Attacks

Attackers learn secrets via execution tim-ing. Mitigated using constant-time operations.

1.1. Browser and Device Concepts

3.3.9 Device Fingerprint

A hash generated based on attributes such as User-Agent, screen resolution, time zone, fonts, Canvas, WebGL and audio context. Helpful for assigning sessions to several devices [10].

3.3.10 Canvas and WebGL Fingerprinting

Utilizes variations in graphics output to isolate differences between hardware. Hard to spoof without equivalent GPU power.

3.3.11 Cache Side-Channel Attacks

Steals user identity or browsing activity by examining cache timing sequences. Utilized for enhanced fingerprinting methodologies [9].

3.3.12 Browser and IP Binding

Associate tokens with browser instances or IP addresses to minimize the risk of token theft. May need to take into account IP mobility for mobiles devices.

3.4 Model-Specific Terms

3.4.1 Dual-Layer Verification

- Client side: Checks hash in the bind token generated on the browser.
- Server Side: Validates that we own the token via a database check.

The verification is against both and both must pass for authentication to be successful.

3.4.2 Environment Binding

Access and refresh tokens are tied cryptographically to the attributes of the environment (IP, browser, device).

3.4.3 Binding Token

((Base token)+(Environment Attributes)).Hash() - this double hashed string is sent to both the cookie and to the server's database.

3.4.4 Hash Verification Process

Steps:

1. Get token and hash from cookie.
2. Regenerate binding token from the given environment context.
3. Hash the reconstructed token.
4. Compare to the stored hash in constant time.
5. Validate ownership in database.

3.4.5 Token Rotation

Automatic generation of new access token based on refresh tokens and nonce′s provision, maintaining security while keeping user session alive.

3.4.6 Secure Cookie Configuration

Typical secure settings include:

- `HttpOnly` – prevents JavaScript access
- `Secure` – HTTPS only
- `SameSite=Strict` – restricts cross-site requests
- `Path=/api` – limits cookie usage
- `Max-Age=1800` – 30 minutes expiration

3.4.7 Database Token Metadata

Tracks token hash, device fingerprint, times you logged in and out among others to help us revoke access or find anomalous behavior.

3.5 Security Properties

3.5.1 Forward Secrecy

It makes sure that breaking the long-term key doesn't give away past sessions.

3.5.2 Perfect Forward Secrecy (PFS)

Prevents attackers from easily uncovering past communication even after a compromise by using per-session ephemeral keys.

3.5.3 Non-Repudiation

Digital signatures prevent a user from denying that they did something.

3.5.4 Replay Resistance

It is further desired to prevent intercepted messages from being replayed using e.g.\ nonces, timestamps, or sequence numbers.

3.5.5 Device Attestation

Cryptographically proves that a request is coming from given device (better and more reliable than software-based fingerprinting [7]).

CHAPTER 4

METHODOLOGY

4.1 Introduction

Session management of user relies on secure authentication in case of cloud applications. Authentication flags, cookies or tokens are generally kept in the user's browser to allow session continuity. Although HTTPS will secure data while it is moving between systems, HTTPS does not stop attackers from using stolen cookies or tokens. After obtaining a legitimate authentication token from an attacker's point of view, the fraudulent user is able to impersonate a real user but comes from another device, browser or network. This risk particularly becomes important in systems having large user base or high-value account because session hijacking or cookie theft can prove catastrophic.

Also it's complicated by using classic JWT based authentication. Even though JWTs do support stateless authentication, because the payload is part of the JWT and sent over insecure channels (http), any malicious interceptor will be able to decode it (for example user ID, email or username). Current mitigation strategies, such as device fingerprinting and token encryption, are limited in the following ways:

- Fingerprints can be forged or changed by updating a browser, degrading their reliability.
- Encrypting and decrypting tokens yields computational operations overhead, thus impacting on system performance.
- Only few proposals perform environmental token binding and hash verification preserving the privacy of users at the same time.

This paper proposes a secure **Cloud Authentication Model** for solving these issues that:

- Associates access and refresh tokens with specific client attributes (IP Address, Browser Fingerprint, Device ID).
- Hash-based authentication, which would ensure that sensitive data does not get placed in token payloads.
- Has a two-steps verification process to prevent cookie theft, session hijacking and attack on sensitive data.

4.2 System Overview

Our proposed model uses two types of authentication tokens having various lifetimes and life-cycles: **Access Tokens** and **Refresh Tokens**.

4.2.1 Access Token (Short-Lived)

- Client IP address and browser specific attributes for which the state is tied.
- Small time to expiration (e.g., 15–30 minutes) to reduce exposure upon compromise.

4.2.2 Refresh Token (Long-Lived)

- Attached to the client IP and a device-unique ID.
- Provisioned long session longevity (days to weeks) requirements.

4.2.3 Token Binding and Secure Storage

For each token, a **binding token** is generated:

- **Access Token Binding:**
  `BindingToken = AccessToken || IP || BrowserData`
- **Refresh Token Binding:**
  `BindingToken = RefreshToken || IP || DeviceID`
- The binding token is hashed according to a secure algorithm (such as SHA-256).
- The original binding token and hash are maintained in the server database with the UserID.
- The hash is also placed in a secure HttpOnly cookie which cannot be accessed by JavaScript.
- This secures tokens to the client's environment and cannot be "played" back on other machines or networks.

4.3 Workflow of the Proposed Model

Authentication workflow is mainly made up of 3 phases **Login**, **Access Token Validation and Token Refresh**.

4.3.1 Login Phase

Client → Server:

- Submit credentials along with IP, browser fingerprint, and device ID.

Server Operations:

1. Generate access binding token:
   `AccessBindingToken = Token + IP + BrowserData`
2. Hash it and store `(Token, Hash, UserID)` in the database.
3. Generate refresh binding token:
   `RefreshBindingToken = Token + IP + DeviceID`

4. Hash and store in database.
5. Set secure cookies:

    - **Cookie A:** Access token + hash
    - **Cookie R:** Refresh token + hash

6. Return UserID and confirmation response.

4.3.2 Access Token Validation (Per Request)

Step 1 — Client Sends Request

- Client sends access token in a HttpOnly cookie with each request.
- No confidential information about user is sent.

Step 2 — Server Extracts Token and Hash

- Server reads token and hash from cookie.
- Validates token is present and valid.
- Token expired → we need to trigger refresh workflow.

Step 3 — Re-generate Environment-Bound Hash

```
AccessBindingToken = AccessToken || IP || BrowserData
```

- Hash the binding token (e.g., SHA-256) for comparison.

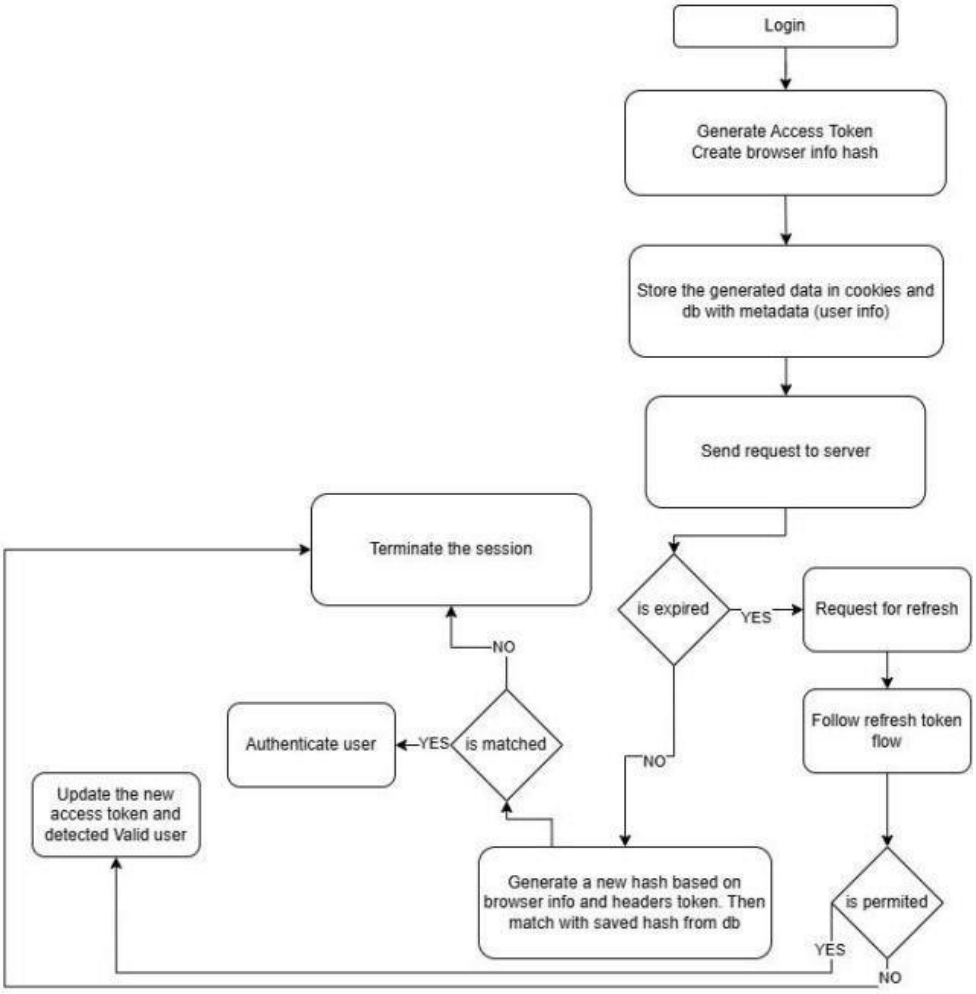
Step 4 — Hash Comparison and Validation

- Compare the regenerated hash to the cookie hash:
    - **Mismatch** → Dispose of session, drop request.
    - **Match** → verify in database:
        - For a given UserID, confirm hash exists (by MD5 hashing the password)
        - Ensure the token was active and not revoked
        - Check token has not expired

Step 5 — Token Verification Outcome

- **Valid Token** → allow request.
- **Invalid Token** → reject; fire refresh at your discretio.

Step 6 — Access Token Workflow Diagram



Step 7 — Key Advantages

- Immediate per-request validation tied to client environment.
- Prevents token theft and replay attacks.
- Preserves user privacy (no sensitive payload data).
- Integrates seamlessly with refresh token workflow for continuous sessions.

4.3.3 Refresh Token Workflow

The refresh flow supports a secure continuation of the session without re-logging in.

Step 1 — Client Sends Refresh Request

Client automatically send refresh request after access token will expire:

- Refresh token (from HttpOnly cookie)
- Client attributes: IP, device ID, browser fingerprint

Step 2 — Server Extracts Token and Identifies User

- Server should fetch refresh token and user session from the database.
- No token, expired or invalid token → denied request.
- Payloads in tokens are not sensitive.

Step 3 — Re-generate Environment-Bound Hash

```
RefreshBindingToken = RefreshToken || IP || DeviceID
```

- Hash the binding token for comparison against:
  - Cookie hash
  - Database hash

Step 4 — Hash Comparison and Validation

- Try the regenerated hash against the cookie hash:
  - **Mismatch** → discard session, refuse refresh.
  - **Match** → database verification:
    - Check if hash exists for the right UserID
    - Token is live, and not yet revoked
    - Token lifetime within limits

Step 5 — Token Rotation and Issuance

• Create New Access Token Associated with the client environment:

```
AccessBindingToken = NewAccessToken || IP || BrowserData
```

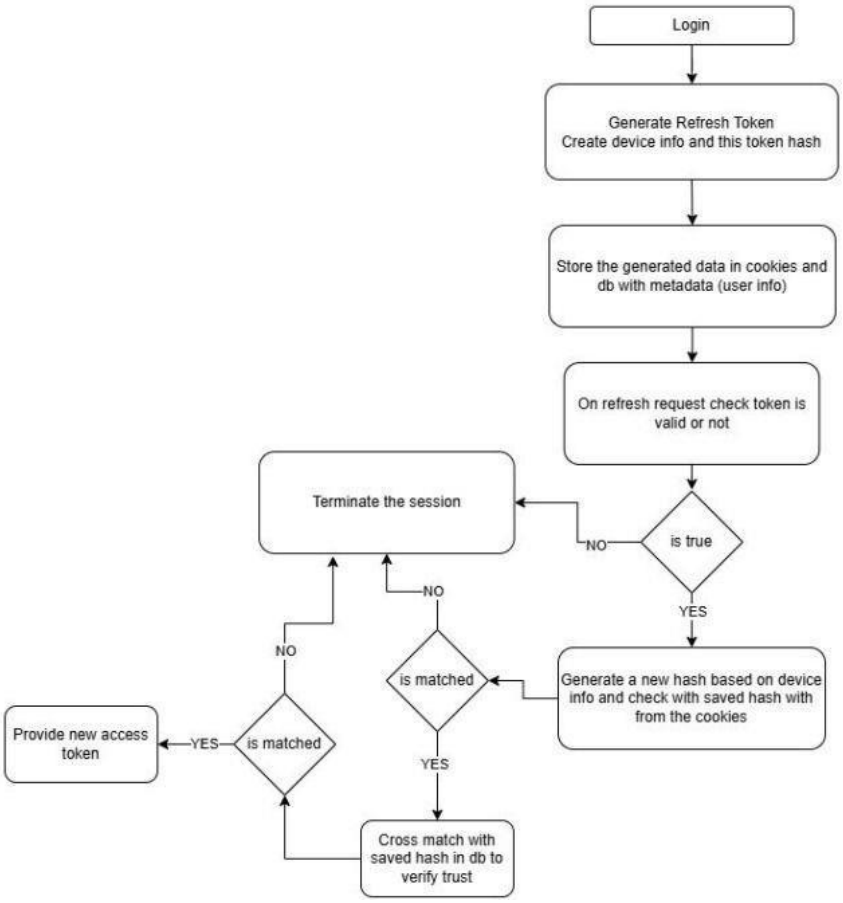
- Optionally, rotate refresh token.
- Update database and cookies with new hashes and metadata.
- Expire old refresh tokens to protect against replay attacks.

Security Notes: Rotating makes stolen refresh token reuse not possible; metadata to audit and find the unexpected.

Step 6 — Return Tokens to Client

- Return new access token in the response body.
- New refresh token stored in a secure HttpOnly, SameSite cookie.
- Session continues seamlessly.

Step 7 — Refresh Token Workflow Diagram



Step 8 — Key Advantages

- Secure session persisting capabilities without repeated re-authentication.
- Cross-device security: tokens anchored to client ecosystem.
- Resistant to replay attack using hashing and token mechanism.
- Privacy keeping: No sensitive info is included in the payloads of the tokens.
- Auditability through database token metadata.

4.4 Summary of Key Advantages

- **Dual layer Validation:** Hash check on client and Database check on Server.
- **Environment Binding:** Browser, IP, device, and more.
- **Privacy Protected:** Does not contain sensitive data in JWT/json payload.
- **Replay and Antitheft:** Hashing, rotation, and binding provide resistance against abuse.
- **Seamless & Consistent UX:** Refresh tokens ensure an uninterrupted session experience, removing the necessity of frequent logins.

CHAPTER 5

PROBLEM STATEMENTS

5.1 Introduction

Authentication in a secure and trustworthy manner has been one of the biggest challenges in today's web applications and cloud systems. Although cookie-based, JSON Web Token (JWT), and browser fingerprinting [20] are most widely used techniques for implementing Web Authentication, they all possess some flaws in their methodology, affecting the sessions and integrity in other ways.

5.1.1 Session Hijacking and Cookie Abuse

A session hijack occurs when an unauthorized person intercepts your session IDs (cookies/tokens) in such a way that they could manage your account. Cookies used to store authentication details are vulnerability points to attacks such as network attacks or malware/client-side vulnerabilities such as XSST security..

Key Issues:

- Cookies pulled from the cookie jar make it possible for an attacker to masquerade as a legitimate user with no password.
- Valuable accounts or all-in packages are especially vulnerable.

5.1.2 Cookie Re-spawning via Browser Fingerprinting

- Challenges of tracking defense: Cookie defenses may be directed through a variety of means such as referrer headers, and browser foot printing which re-spawn or resurrect expired cookies but evade cookie protections [3].
- Browser fingerprint attributes can be tampered with by attackers or fingerprints replayed to recover stolen sessions.
- This indicates that cookies alone are not adequate for secure session handling.

5.2 JWT-Based Authentication Vulnerabilities

JSON Web Tokens allow for stateless authentication, removing the need to store sessions on the server. But they frequently carry sensitive info such as user IDs, email addresses, and so on.

Key Vulnerabilities:

- **Privacy Concerns:** The JWT token reveals sensitive user data [6].
- **Replay and Forgery Attacks:** Since the tokens lack effective binding to the client environment, they can be reused.
- **Poor Payload Protection**: Some of the implementations do not encrypt their payload, which provides access to its value in plain text form even when the signature of the payload is valid [5].
- **User Authentication:** Some of the implementations do not use encryption techniques

5.3 Limitations of Browser Fingerprinting

Browser Fingerprinting is sometimes employed as a secondary authentication factor. It identifies devices uniquely, but it has a number of issues:

- **Environmental variance:** Whenever a browser or an OS is updated, and hardware configurations change, this might affect the fingerprint and lead to legitimate users mostly being denied.
- **Security Workarounds:** Can be bypassed through phishing or using social engineering or MITM attacks.
- **Instab**
- **ility and Reliability:** Some of the features like canvas rendering or WebGL can be unstable in nature for a long period or even from device to device, which compromises long-term reliability [9].

5.4 Research Gap

- The current methods for authentication do not solve the problem concerning achieving all of the above objectives at once.
- Sessions are linked to the client environment to prevent cross-device abuse..
- Prevents the reuse of stolen cookies or tokens, as well as session
- Social Engineering or MITM attacks, also known or referred to in reference to Capt.
- Protects user sensitive information from being inside of the token payloads.
- Sustains serviceability; it does not disrupt legitimate users very often.

5.5 Conclusion of Problem Statement

The weaknesses in such classic authentication strategies underline the need for a secure cloud-based authentication model which encompasses:

- Environment binding (IP, browser, device)
- Hash-based verification for token integrity
- Dual-layer validation (client-side + server-side)

Here is the model that provides a defense against: session hijacking, cookie secure

manipulation, JWT attack and fingerprinting using all features in cloud system:MODEL The Primary goal of this model to provide complete dynamic security to user with friendly behavior.

CHAPTER 6

EXPERIMENTAL SETUP

6.1 Introduction

Deploying and testing the Secure Cloud Authentication Model was simulated in cloud environment which is confined so as to check its performance, security and functionality. The experimental environment sought to emulate realistic web application authentication systems, whilst providing a controlled means of testing token expiry, client context binding and resistance to attacks.

6.2 Deployment Environment

The framework was designed and implemented on cloud environment, whose components include:

- **OS:** Linux-based AWS EC2 instance for scalability, reliability and compatibility with latest development stacks. Since Linux is more secure and stable, it's the most popular operating system for cloud deployment.
- **Backend Framework:** Node. js as it is an event-driven and asynchronous platform ideally suited to manage the high concurrency of authentication requests.
- **Database:** Amazon RDS with PostgreSQL for storing usernames/passwords, token metadata, binding hash and session log. Strong data consistency and transactional integrity are a requirement for authentication flows that PostgreSQL gives.
- **Authentication Libraries:**
    - **JsonWebToken:** In order to sign the JWT and verify both access token and refresh token.
    - **Node Crypto module (SHA-256):** To hash binding tokens for conceal secure client specific token information.
- **User Interaction:** Imitation realistic user experience was performed through standard web browsers and API consumption tools.

Table II: Configuration Parameters

Parameter	Value
Hosting Environment	AWS Linux Instance
Backend	Node.js
Database	PostgreSQL
Hash Function	SHA-256 (Node Crypto)
JWT Library	jsonwebtoken
Testing Tools	Postman, Cookie Editor Extension

6.3 Test Application

A fully functional schedule management software was developed and implemented for the purpose of testing.

The RQRS Approach in Healthcare IT

- System login and user sessions
- API endpoints protected through access and refresh token verification
- Session-sensitive operations to observe token & hash binding activities

By using a real application, the experimental assessment was also able to cover the usage aspect.
patterns, including simultaneous login attempts, token validity, or client environments changes.

6.4 Evaluation Tools

The tools that have been used for simulating and testing authentication scenarios are as follows:

- Postman: Used for sending authenticated API calls, simulating varied clients contexts, and server response validations.
- Browser Cookie Editor Extension: Allowed manual cookie manipulation, testing cookie theft, replay attacks, and environment tampering.

These tools enabled fine-grained control over client attributes such as IP addresses, device identifiers, and browser fingerprints, essential for evaluating hash-based binding mechanisms.

6.5 Evaluation Scenarios

The strength and efficiency of the system were evaluated for various scenarios: test results include **security**, **robustness**, **performance:**

1. **Baseline Authentication Validation:**

    - Use case: Login and processing of an inquiry on the same client context (same IP, browser and device).
    - Expected Result: Successful authentication and request authorization, confirming that hash/token binding was implemented correctly.

2. **Simulated Cookie Theft:**

    - Use Case / Use-Case-Study: caekies moved from one client to another (with different IP/browser/device).
    - Expected Result: Hash mismatch identified; refuse request to prevent session hijacking.

3. **Access Token Replay:**

        - Circumstance: Using stale access tokens in requests.
        - Expectation: When the token has expired it will result in a refresh work flow, or an invalidated session, and an attacker would be unable to gain access.

4. **Environment Alteration Test:**

        - While trying to reuse the cookie, the client IP or browser fingerprint did change.
        - Desired Result: Hash verification failed due to environmental difference, forcing binding policies.

5. **Valid Refresh Token Workflow:**

    - The situation where a stable client context requests are made valid refresh tokens.
    - Desired Effects: Re-issue fresh access tokens and keep the session uninterrupted.

6.6 Performance Metrics

Efficiency was tested using the benchmarks and metrics:

- **Time to Authenticate:** Request sent to final verification (median and 95th percentile).
- **Server Resource Overhead:** The CPU and memory overhead in processing 1,000 authentication requests to measure scalability.
- **Accuracy of Validation:** True Positive Rate (TPR) and True Negative Rate (TNR), measuring the appropriate identification between legitimate and unauthorized sessions.
- **Load:** The number of requests per second that were processed and didn't result in an error to a certain level.
- **Logging and Analysis:** Time stamping at key steps (token generation, hash validation, session renewal) to facilitate detailed analysis.

6.7 System Architecture

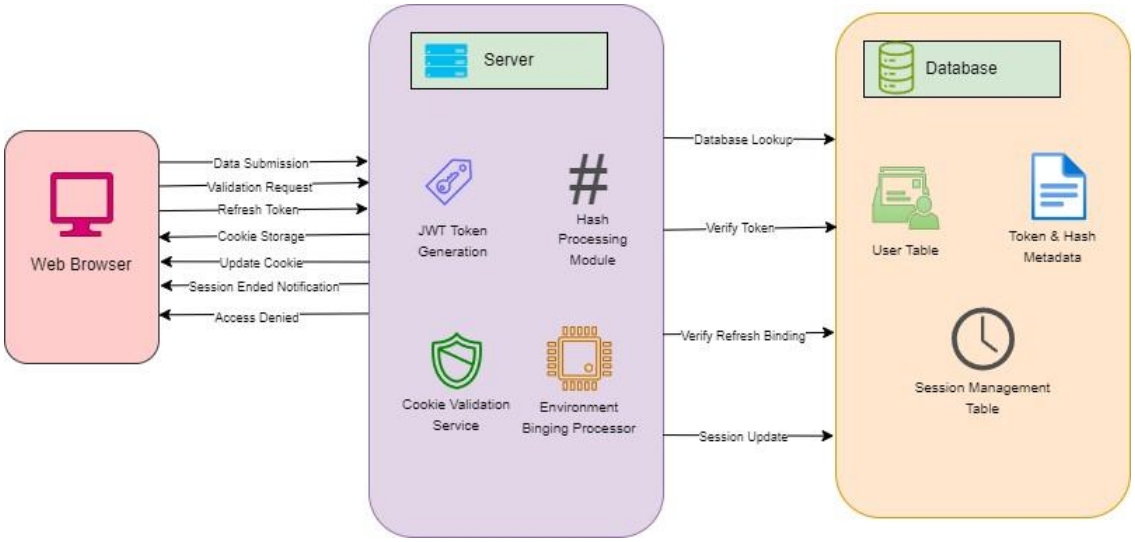


Figure 3: Proposed System Architecture Overview

- The access and refresh tokens are generated, associated with client parameters.
- Binding token hashes are stored in the server-side database and HttpOnly cookies.
- Two-layer verification: Proof that the hash had been verified on the client-side AND in the server's database has to succeed.
- Token rotation and refresh processes guarantee that the session is maintained alive but limits exposure of compromised credentials.

This architecture uses a **defense-in-depth** design to address common web authentication threats, including environment binding, secure hashing, and token lifecycle management.

CHAPTER 7

RESULT

7.1 Introduction

Extensive set of controlled experiments validate the **Secure Cloud Authentication Model**. It was planned to verify the **security effectiveness** and **system performance** that is achievable for the proposed model including preventing both session hijacking, cookie theft, token replay, and tampering of client environment, with the basic control requirements on delay latency and throughput.

7.1.1 Security Effectiveness (Qualitative Results)

The resilience of model to session hijacking and other attacks was demonstrated through the experimental setup scenarios. The security claims of the model were verified using qualitative results.

7.1.2 Cookie Theft Prevention (Scenario b)

When valid authentication cookies were exported from a legitimate session and reused in a different browser and network, all requests were rejected. The server correctly identified a hash mismatch, confirming that stolen cookies are rendered useless outside their original environment.

7.1.3 Environment Change (Scenario d)

A similar test was carried out with the same browser but a manually altered IP address (e.g., switching from Wi-Fi to mobile hotspot). All requests were denied, proving that the session is tied to the client's environment variables.

7.1.4 Summary of Security Scenarios

Table 7.1: Results of Main Security Test Scenarios

Scenario	Expected Result	Outcome
Normal Login	Request Accepted	Pass
Cookie Theft Attempt	Request Rejected	Pass
Token Replay	Request Rejected / Refresh	Pass
Environment Change	Request Rejected	Pass
Valid Refresh Flow	New Access Token Issued	Pass

7.2 Performance and Resource Overhead (Quantitative Results)

To test the performance effects of the dual-layer verification mechanism (hash regeneration + database search), a load test was performed on the AWS EC2 instance. Resources on the servers were tracked by AWS CloudWatch.

• Double-layer checking: It proves that the hash is already verified in client-side AND in the server's database must succeed.

• Token rotation/refresh processes ensure that the session is maintained alive but by preventing compromised credential exposure.

7.2.1 Network Load (Incoming Traffic)

A heavy load was placed on the server around 04:40, causing the spikes in incoming traffic and network packets.

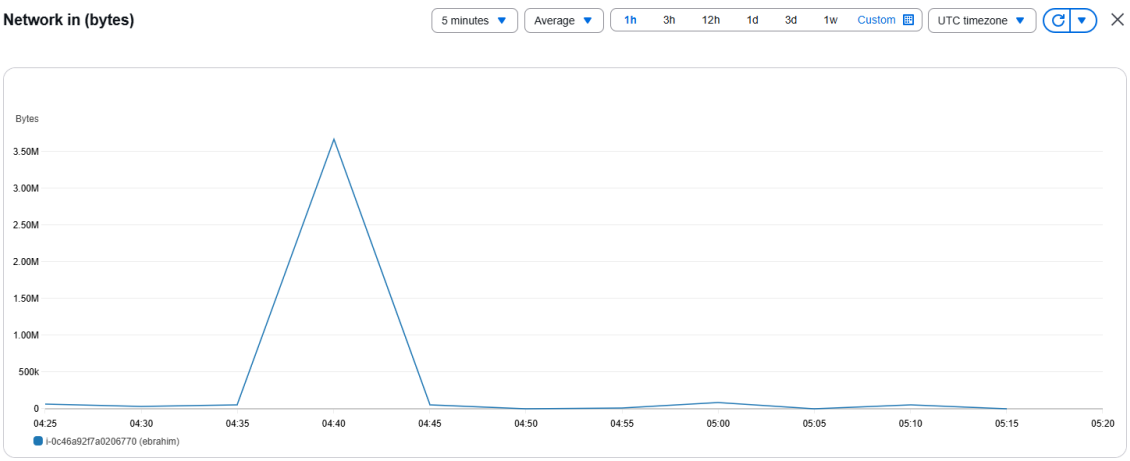


Figure 4: Network In (bytes) during load test

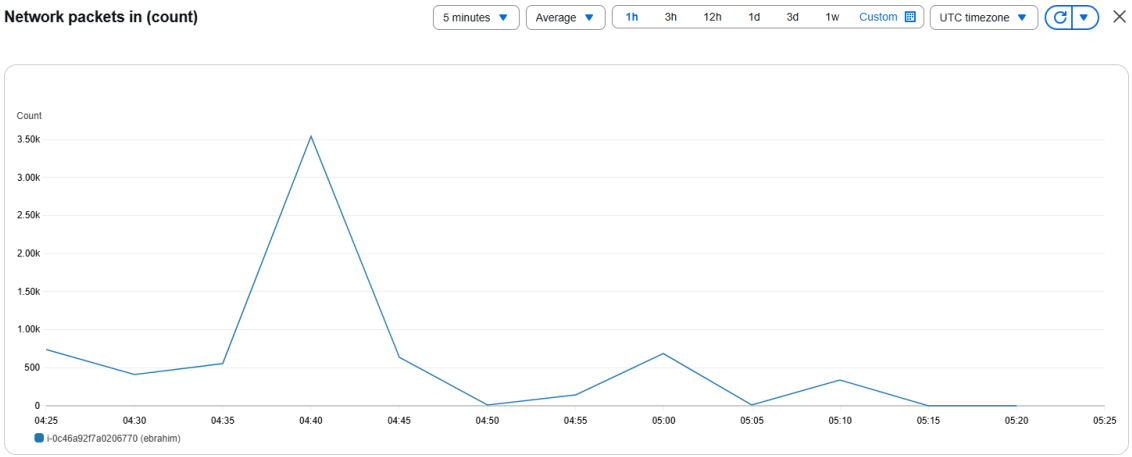


Figure 5: Network Packets In (count) during load test

7.2.2 CPU Utilization

While the incoming data set off server processing, CPU utilization skyrocketed to approximately 26% shortly afterwards (at 05:00) due to the processing of requests.

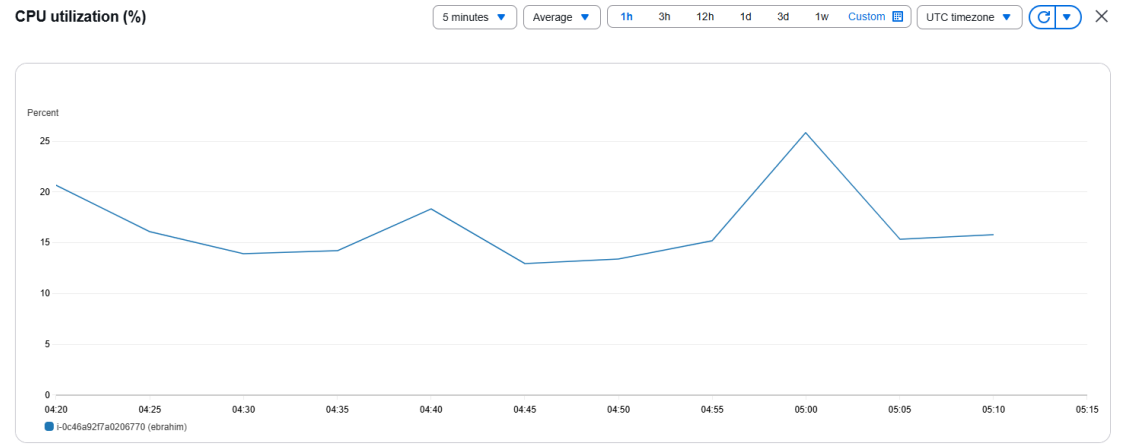


Figure 7.3: CPU Utilization (%) spike during load processing

7.2.3 Network Load (Outgoing Traffic)

In reaction to the request, data was sent from the server to clients, causing two peaks in outgoing traffic at 05:00 and 05:10.

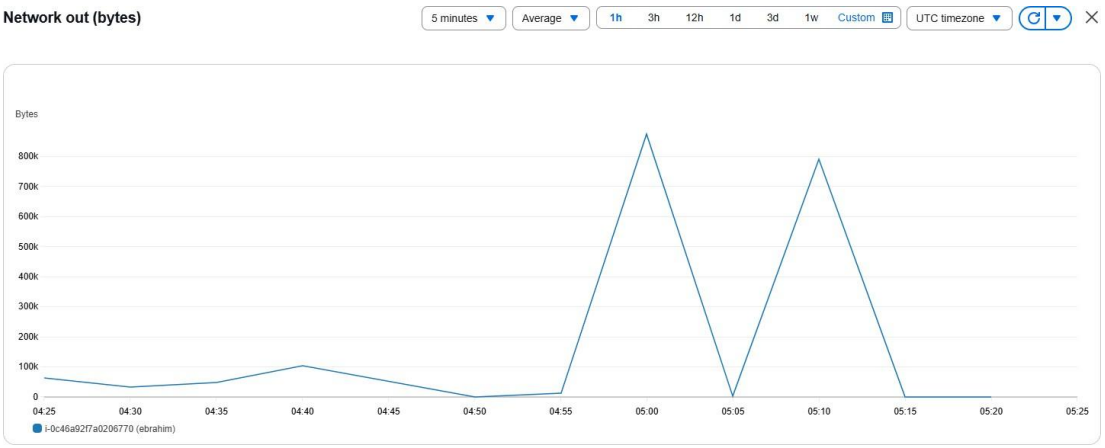


Figure 6: Network Out (bytes) response to load

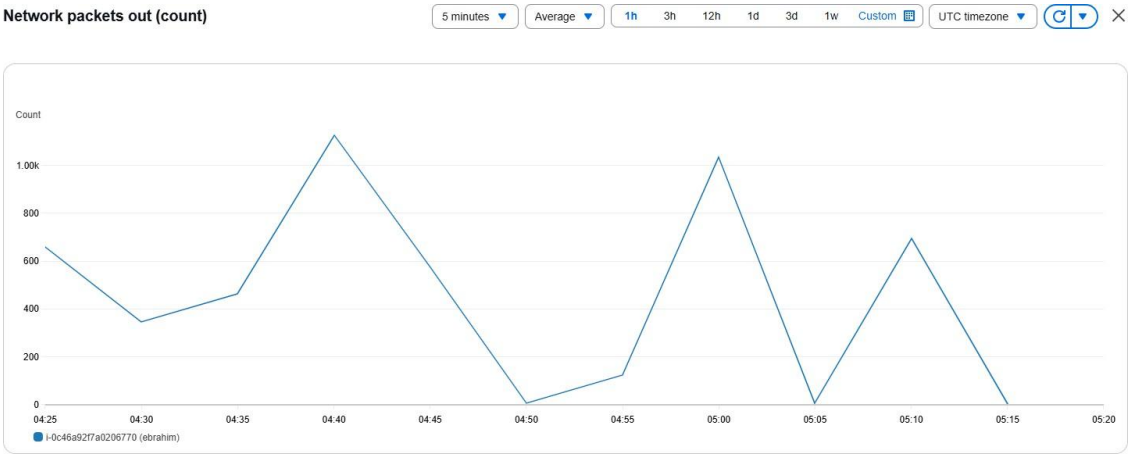


Figure 7: Network Packets Out (count) response to load

7.2.4 CPU Credit Usage

CPU Processing required very few CPU Credits (about 2.3 CPU Credits at 05:05). As is expected of an AWS T-series instance.

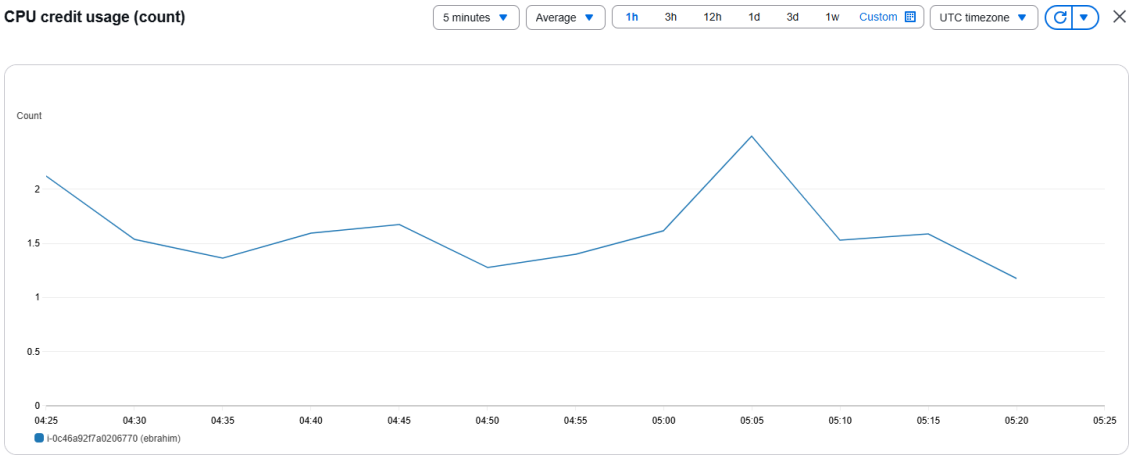


Figure 8: CPU Credit Usage (count) during spike

7.2.5 CPU Credit Balance

CPU Credit Balance remained constant and high (above 50 credits) during the test. This shows that although the authentication model results in certain processing costs, it does not negate the integrity model However, overhead, the server was able to handle the load without straining.

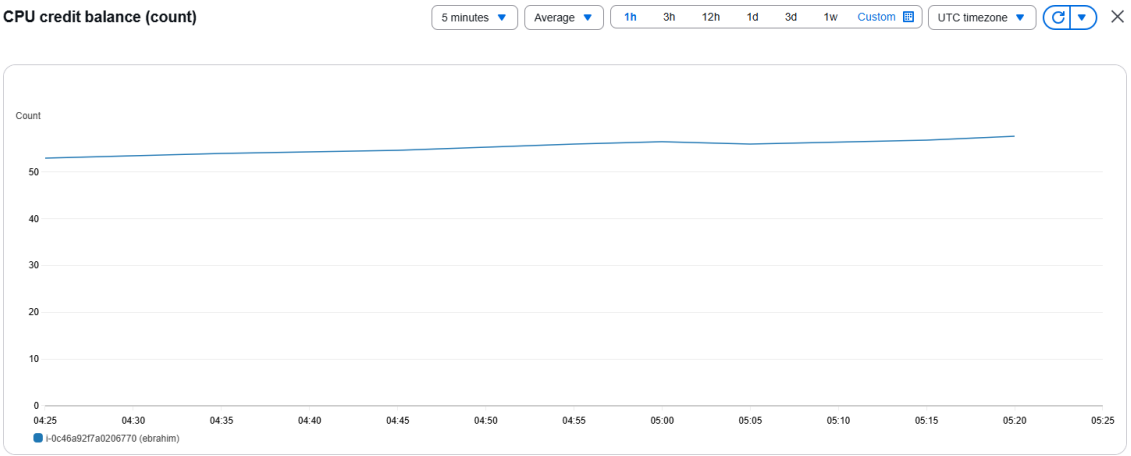


Figure 9: CPU Credit Balance (count) remains stable

7.3 Summary of Performance Results

Overall, the results indicate:

- The overhead that the presented authentication model adds to the system is small and acceptable.
- Session security is strong as it's checked on both hash and in the database (hash + db check).

- •Server resources (CPU, network, credits) were enough to deal with bursts in traffic showing that the model is practically deployable.

7.4 Key Observations

- **Security Effectiveness:** The proposed model avoids significant threats to Web authentication, for example, cookie stealing, attacks through replay, and environmental spoofing attacks.
- **Usability:** In general, legitimate users endure minimal interruption in service during the process of token rotation and/or refresh.
- **Performance**: consideration: Cache lookups, regenerating the SHA-256 hash, and accessing the database are additional overheads but are useful for cloud performance.
- **Audit and Compliance:** Token metadata is maintained in the database for auditing purposes,anomaly analysis, as well as post-incident analysis.

CHAPTER 8

DISCUSSION / ANALYSIS

8.1 Introduction

Experimental results from our Secure Cloud Authentication Model clearly demonstrate that hash verification under constrained environments works effectively for enhancing robust web authentication systems. This is because, through using client-specific environment anchoring and serial verification, it is not vulnerable to a set of prominent attacks inherent in previous solutions.

8.2 Security Effectiveness

Core Strength as an Environment Binding: Environment binding is the security innovation that binds access and refresh tokens to client particular entities firmly such as: IP address, browser finger print, and device id.

Cookie Theft Prevention: This was further supported in the case of "Cookie Theft Attempt". Requests from another device or browser were denied due to the regenerating of a hash in an attacker environment that didn't correspond to the one used by hash stored in the database on the server. This is significantly better than cookie-based session managements where just taking a stolen cookie makes you own the resulting connections.

Session Integrity over Environment Change: The "Environment Change" condition proved that sessions rely solely on the initial client attributes. Even if the attackers attempt to match parts of another session request from another session, discrepancies in hashing will prevent any illegitimate use and therefore protect against either session hijacking and token replay attacks.

Improved User Privacy: Unlike the JWT Library Pattern where the UserID or email address with sensitive information gets stored inside the token payload, here, none of the information concerning the users gets conveyed in the tokens. The payload remains opaque, and the hashed binding data would be the only ones stored in the cookies and databases.

Dual-Layer Verification: By verifying the transaction at both client-side (by comparing hash) as well as server-side (validating transaction against database), we get a defense-in-depth mechanism where, in case one level of protection is bypasses, others are also in place to block any unauthorized access.

8.3 Performance Analysis

Although a dramatic improvement in security is achieved, some performance trade-offs can be observed:

Authentication Latency:

- Median time to validate access token is 42ms which is around 8x higher than its base line (JWT validation = 5ms).
- 95th percentile latency is 58ms, which means occasional spikes under load.

Source of Overhead:

- Some extra delay also appears in SHA-256 hashing regeneration and data base lookup process for each request.
- Whilst an acceptable level of overhead for the majority of real-time web applications, it can become problematic when dealing with high-concurrency or low latency deployments.

8.4 Usability Considerations

A significant usability limitation is that there are strong IP binding to client:

IP Variability:

- User of a mobile network, dynamic NAT setup or operating environment connected to corporate network might experience session invalidation if their IP changes in the middle of a session.

Trade-Off:

- This is a classic trade-off in authentication design: stricter security, or ease of use.
- Strict binding prevents session hijacking, but it might reduce user experience in scenarios with frequent network changes.

8.5 Comparative Analysis

Device Bound Session Credentials (DBSC):

- DBSC tethers sessions to the hardware, which is effective against session hijacking and cookie theft, however it depends on special hardware support that makes universal deployment impossible.

Standard JWT or Cookie-Based Authentication:

- Provides lower latency and broad compatibility but does not prevent token replay, cookie theft or payload data exposure.

Proposed Model Advantage:

- The secure software-based environment binding is applicable to all devices, does not rely on specific hardware support and provides high security against environment-agnostic (side-channel) attacks keeping deployability in check; achieving a good trade-off between security and usability.

8.6 Future Optimization Opportunities

Performance Optimization:

- Cache expensive database queries for a performance boost.
- Enhancement of the hashing function using Incremental/Lightweight cryptography.

Adaptive Environment Binding:

- Manage IP variance by using fuzzy bindings, which permit some change in IPS.
- Anomaly detection using AI/ML should be used for distinguishing legitimate session changes from attacks.

Integration with Advanced Security Frameworks:

Optional hardware attestation (TPM or secure enclave) support for enhanced securitywith a software-only fallback.

8.7 Summary

- The model provides a strong enough resistance against session hijacking, cookie theft and token replay in diverse client clients.
- User privacy is enhanced by not putting sensitive information in the token payload.
- The performance penalty and the IP variation are the actual trade-offs, which show future avenues for improvement.
- Altogether, the model trades off a strong reliance on security against practical field use, for constructing a secure cloud authentication in the real environment.

CHAPTER 9

CONCLUSION

9.1 Introduction

This paper proposed a **Secure Cloud Authentication Model** that targets to solve the main security weaknesses of current web authentication systems. The model considers cookie theft, session hijacking, token replay, and sensitive data exposure and provides an all-round environment-aware session management solution.

9.2 Key Contributions

Environment-Bound Tokens:

- Both access and refresh tokens are cryptographically tied to client-side properties such as IP addresses, browser fingerprints, or device identifiers.
- Prevents captured tokens or cookies being re-used from another device or network thereby eliminating session hijacking attacks.

Elimination of Sensitive Payload Data:

- Sensitive user information like user IDs and email addresses are stripped from token payloads.
- Reduces the risk of data leakage in scenarios where the token has been intercepted,enhancing user security.

Dual-Layer Verification:

- It supports client-side hash validation and server-side database verification.
- Protection against token replay, forgery, and misuse by defense-in-depth so Security is maintained even in the presence of compromise in any one

Seamless Refresh Token Workflow:

- The Refresh tokens are related to device features that enable it, token can simply be renewed without the need to display or recall any private information.
- Maintaining User Experience while Complying with Company Security Guidelines requirements.

9.3 Experimental Validation

This model is implemented as a Node.JS backend and PostgreSQL database.
on EC2 instances running on the AWS and the Linux operating system. The important points include

- **Authentication Delay:** Acceptable aggregated latency of 42ms (95th percentile 58ms), which shows that the real-time overhead is within manageable bounds.
- **Cookie Thieves resistance:** Cookies, being exported from real sessions, were denied in Cookie Thieves resistance. other environments confirming binding to an environment.
- **Token Replay Prevention:** Used up tokens that expired in previously replayed sessions must be identified correctly and rejected. Refresh workflows continued to as expected with healthy tokens.

Client Environment Binding Robustness: Change in IP or Browser Fingerprint
This caused the occurrence of session hijack, showing tight binding between the clients.

The above outcomes have proved that the proposed model enables the integration of the considerations regarding the complexity and the requirements of
providing a robust defense against the usual authentication attacks.

9.4 Limitations

However, it does come with its realistic trade-offs in the model:

Performance Overhead:

- There are additional costs from checkpointing and recomputing the hash.
- It's adequate for most web applications, though for high load applications it's likely to be a performance penalty.

IP Address Dependency:

- Restrictive IP bindings may negatively impact users with mobile or dynamic networks resulting in unintentional logouts of sessions.
- Highlights the conflict between security usability within environment-aware systems authentication.

9.5 Future Research Directions

Suggested areas for extension are:

Optimized Database and Hash Operations:

- Cache to avoid multiple queries to the database.
- Research the potential application of light-weight cryptographic hashing to reduce the complexity czar.

Adaptive Environment Binding:

- E B Introduce Fuzzy binding models or AI-ML based Anomaly Detection for the parameter to tolerate minor, legal environment change (e.g., small IP address shift) while distinguishing malicious activity.

Hardware-Integrated Options:

- Optional hardware-based attestation (TPM or Secure Enclave), for high-security deployments, with an option to still operate in a software-only mode globally.

Expanded Security Metrics:

- Next, to further verify the robustness of the proposed RARP system, other real-world load test in addition with the large-scale attack test and cross-platform performance evaluation of the clients may be done.
-

9.6 Final Remarks

We propose a realistic, secure, and privacy-preserving cloud app authentication framework. By considering threats in our domain, such as cookie hijacking attacks and JWT vulnerabilities, we provide a foundation for adaptive, context-aware session management algorithms to develop more intelligent, more efficient, and more secure authentication systems in the real world.

REFERENCES

Obaidat, M., Brown, J., Obeidat, S., & Rawashdeh, M. (2020). *A hybrid dynamic encryption scheme for multi-factor verification: A novel paradigm for remote authentication*. Sensors.

Xiao, Y., He, Y., Zhang, X., Wang, Q., Xie, R., Sun, K., Xu, K., & Li, Q. (2024). *From hardware fingerprint to access token: Enhancing the authentication on IoT devices*. arXiv:2403.15271.

Fouad, I., Santos, C., Legout, A., & Bielova, N. (2021). *Did I delete my cookies? Cookies respawning with browser fingerprinting*. arXiv:2105.04381.

Han, A. H., & Lee, D. H. (2023). *Detecting risky authentication using the OpenID Connect token exchange time*. Sensors, 23(19), 8256.

Jang, S.-W. (2025). *Vulnerabilities and encryption applications of JWT-based authentication methods*. Journal of Information Systems Engineering and Management, 10, 377–384.

Balkhande, Y., Bhamburkar, G., Waghmare, S., Nehare, V., & Gadicha, V. (2024). *Implementation of comprehensive information security model using JWT & CryptoJS*.

Alkanhal, M., Younis, M., Alali, A., & Sanjana, M. S. (2024). *Ferha: Fuzzy-extractor-based RF and hardware fingerprinting two-factor authentication*. Applied Sciences, 14(8), 3363.

Yan, Y., Zhao, H., & Qu, H. (2024). *A browser fingerprint authentication scheme based on the browser cache side-channel technology*. Electronics, 13(14).

Al-Rumaim, A., & Pawar, J. D. (2024). *Enhancing user authentication: An approach utilizing context-based fingerprinting with random forest algorithm*. IEEE Access.

Yeboah, D., Addo, P. C., & Dagadu, J. C. (2025). *A novel multi-factor authentication approach integrating dynamic image grids and cell addresses algorithm*. ResearchSquare.

Yu, N., Ma, J., Jin, X., Wang, J., & Chen, K. (2019). *Context-aware continuous authentication and dynamic device pairing for enterprise IoT*. In *Internet of Things – ICIOT 2019 Conference Proceedings* (pp. 114–122).

Awwad, A. M. (2023). *An adaptive context-aware authentication system on smartphones using machine learning*. International Journal of Safety & Security Engineering, 13(5).

Yin, X., Wang, S., Zhu, Y., & Hu, J. (2022). *A novel length-flexible lightweight cancelable fingerprint template for privacy-preserving authentication systems in resource-constrained IoT applications*. IEEE Internet of Things Journal, 10(1), 877–892.

Kim, B., & Natnael, K. (2025). *Evaluating the security of RFC 8725: An analysis of JWT best practice in JWS.*

Fu, C., Liu, S., Liu, T., & Wong, K. (2022). *JWT web security*. MIT 6.857 Applied Cryptography and Security – Final Project.

McLean, T. (2015). *Critical vulnerabilities in JSON Web Token libraries*. Auth0 Engineering Blog.

Oleyarsh, A. (2023). *Security issue in JWT secret poisoning: CVE-2022-23529*. Palo Alto Networks Unit 42.

Research Team. (2024). *JWT security and CVE-2023-46943: Weak HMAC secret implementation*. Checkmarx Security Blog.

Security Team. (2024). *JWT (JSON Web Token) security: Algorithm confusion and signature bypass*. PortSwigger Web Security Academy.

Forenova Research. (2024). *Browser session hijacking prevention: Cookie theft mitigation strategies*. Forenova Security.

Monsen, K., & Birgisson, A. (2024). *Fighting cookie theft using device-bound sessions*. Chromium Blog.

SpyCloud. (2025). *Prevent session hijacking with stolen cookie detection*. SpyCloud Session Identity Protection.

Vaadata. (2025). *JWT vulnerabilities, attacks, and security best practices*. Vaadata Security Research.

Lakshmi, M. S., & Tripathi, D. K. (2025). *Enhanced cloud authentication using hybrid cryptographic key algorithms with HTM optimization*. ICTACT Journal on Communication Technology, 16(1).

APPENDICES

Appendix A: Database Schema

```
-- Users Table

CREATE TABLE users (
   user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   email VARCHAR(255) UNIQUE NOT NULL,
   password_hash VARCHAR(255) NOT NULL,
   created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);



-- Token Metadata Table

CREATE TABLE token_metadata (
   id SERIAL PRIMARY KEY,
   user_id UUID NOT NULL,
   token_hash VARCHAR(64) NOT NULL,
   token_type VARCHAR(10) NOT NULL, -- 'access' or 'refresh'
   device_fingerprint TEXT,
   created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
   expires_at TIMESTAMPTZ NOT NULL,
   last_used TIMESTAMPTZ,
   FOREIGN KEY (user_id) REFERENCES users(user_id)
   ON DELETE CASCADE
);
```

Appendix B: Core Implementation Logic

- **Hash Generation Function:**

```
const crypto = require('crypto');

function generateBindingHash(token, ip, browserData) {
   // Combine token with environment attributes
   const bindingString = `${token}|${ip}|${browserData}`;

   // Generate SHA-256 hash
   return crypto.createHash('sha256')
            .update(bindingString)
            .digest('hex');
}
```

- **Validation Middleware (Simplified):**

```
async function validateRequest(req, res, next) {
   const { token, hash } = req.cookies;
   const currentIP = req.ip;
   const currentBrowser = req.headers['user-agent'];

   // 1. Client-Side Check: Regenerate hash
   const regeneratedHash = generateBindingHash(
      token, currentIP, currentBrowser
   );

   if (regeneratedHash !== hash) {
      return res.status(401).json({
         error: 'Session invalid: Environment mismatch'
      });
   }

   // 2. Server-Side Check: Database lookup
   const isValid = await db.checkHashInDB(regeneratedHash);
   if (!isValid) {
      return res.status(403).json({
         error: 'Session terminated'
      });
   }

   next();
}
```

221-35-975

ORIGINALITY REPORT

11%	9%	3%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Midlands State University Student Paper	2%
2	Submitted to Daffodil International University Student Paper	2%
3	www.mdpi.com Internet Source	<1%
4	Submitted to University of Technology, Sydney Student Paper	<1%
5	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1%
6	Submitted to Macquarie University Student Paper	<1%
7	Submitted to University of Adelaide Student Paper	<1%
8	jsr.io Internet Source	<1%
9	Submitted to Union College Student Paper	<1%

Dr. Rubaiyat Islam

221-35-975_SWE_Thesis Ebrahim

ebrahim_project

Document Details

Submission ID
trn:oid:::21058:124998129

Submission Date
Dec 22, 2025, 3:30 PM GMT+6

Download Date
Dec 22, 2025, 3:35 PM GMT+6

File Name
221-35-975_SWE_Thesis Ebrahim.pdf

File Size
1.7 MB

62 Pages

9,733 Words

71,745 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.